操作系统期末大作业

学　院：智能工程学院

姓　名：刘德鹏

学　号：18364059

邮　箱：liudp5@mail2.sysu.edu.cn

提交日期：2021/01/24

Xv6 lab: Multithreading/Uthread: switching between threads

仿照上下文切换的方式完成进程切换
在 trampoline.S 中的形式为
首先修改 uthread.c 中的 thread 定义如下

```
struct thread {
  /** My Implementation  */
  uint64      ra;
  uint64      sp;
  // callee registers
  /** thread_switch needs to save/restore only the callee-save register
s.   */
  uint64      s0;
  uint64      s1;
  uint64      s2;
  uint64      s3;
  uint64      s4;
  uint64      s5;
  uint64      s6;
  uint64      s7;
  uint64      s8;
  uint64      s9;
  uint64      s10;
  uint64      s11;

  char      stack[STACK_SIZE]; /* the thread's stack */
  int         state;             /* FREE, RUNNING, RUNNABLE */
};
```

模仿 proc.h 中进程上下文的切换，在 uthread_Switch.S 中补充线程上下文的切换：
修改 uthread_Switch.S 为

```
	.text

/* Switch from current_thread to next thread_thread, and make
 * next_thread the current_thread.   Use t0 as a temporary register,
 * which should be caller saved. */

	.globl thread_switch
thread_switch:
	/* YOUR CODE HERE */
	sd ra, 0(a0)
	sd sp, 8(a0)
	sd s0, 16(a0)
	sd s1, 24(a0)
	sd s2, 32(a0)
	sd s3, 40(a0)
	sd s4, 48(a0)
	sd s5, 56(a0)
	sd s6, 64(a0)
	sd s7, 72(a0)
	sd s8, 80(a0)
	sd s9, 88(a0)
	sd s10, 96(a0)
	sd s11, 104(a0)

	ld ra, 0(a1)
	ld sp, 8(a1)
	ld s0, 16(a1)
	ld s1, 24(a1)
	ld s2, 32(a1)
	ld s3, 40(a1)
	ld s4, 48(a1)
	ld s5, 56(a1)
	ld s6, 64(a1)
	ld s7, 72(a1)
	ld s8, 80(a1)
	ld s9, 88(a1)
	ld s10, 96(a1)
	ld s11, 104(a1)
	ret	/* return to ra */
```

修改 thread_create，使之能够记录线程的返回地址 ra 与栈地址 sp。其中，返回地址意味着当切换线程时，线程应该返回到什么哪个地址。这里应该是传入的函数的入口 func

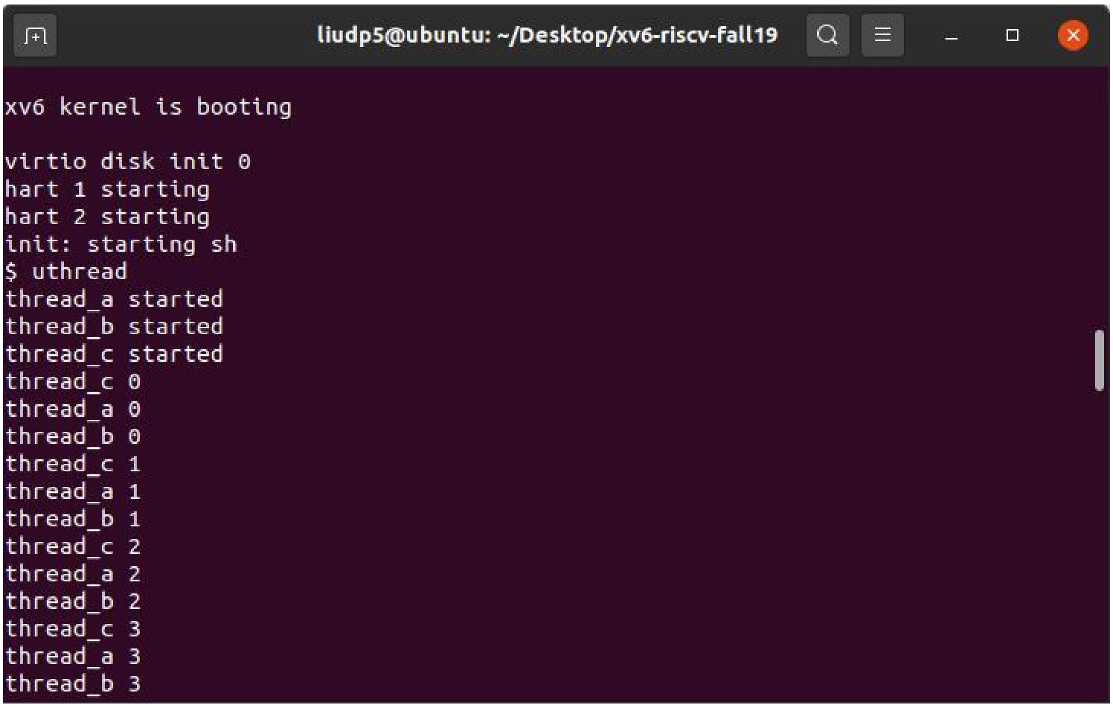
```
void
thread_create(void (*func)())
{
  struct thread *t;

  for (t = all_thread; t < all_thread + MAX_THREAD; t++) {
    if (t->state == FREE) break;
  }
  t->state = RUNNABLE;
  // YOUR CODE HERE
  t->ra = (uint64)func;
  t->sp = (uint64)(t->stack + STACK_SIZE);
}
```

最后，在 thread_schedule 中添加 thread_switch 调用：

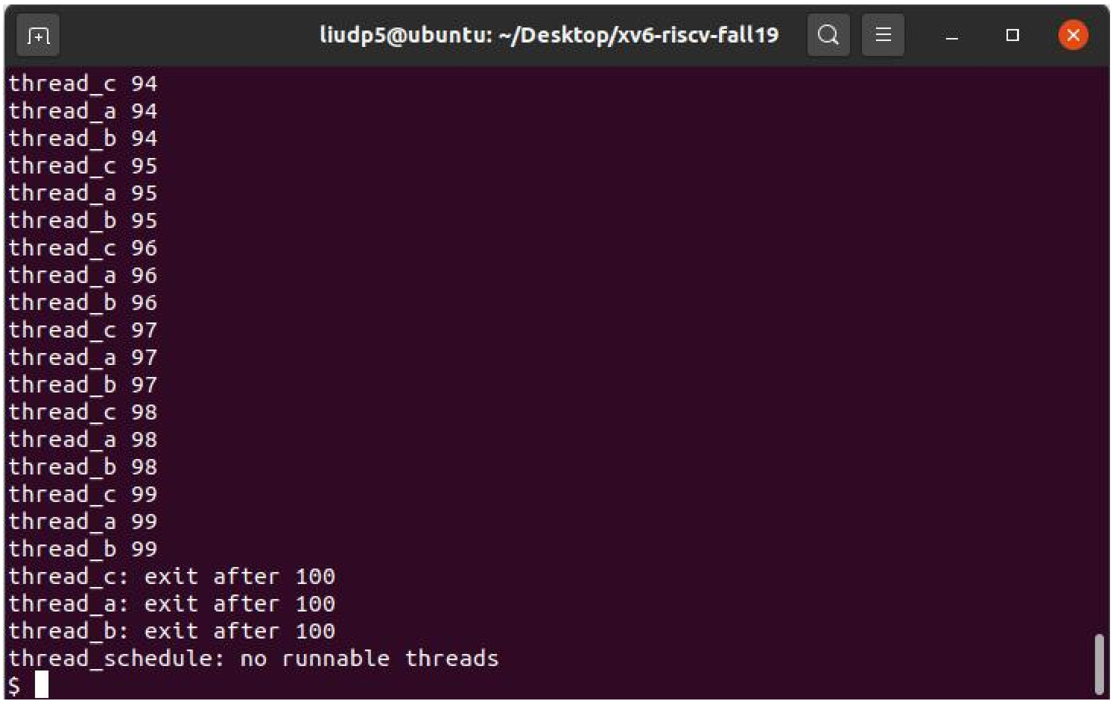
```
void
thread_schedule(void)
{
  struct thread *t, *next_thread;

  /* Find another runnable thread. */
  next_thread = 0;
  t = current_thread + 1;
  for(int i = 0; i < MAX_THREAD; i++){
    if(t >= all_thread + MAX_THREAD)
      t = all_thread;
    if(t->state == RUNNABLE) {
      next_thread = t;
      break;
    }
    t = t + 1;
  }
  if (next_thread == 0) {
    printf("thread_schedule: no runnable threads\n");
    exit(-1);
  }
  if (current_thread != next_thread) {         /* switch threads?  */
    next_thread->state = RUNNING;
    t = current_thread;
    current_thread = next_thread;
    /* YOUR CODE HERE
```

```c
     * Invoke thread_switch to switch from t to next_thread:
     * thread_switch(??, ??);
     */
     thread_switch((uint64)t, (uint64)next_thread);
  } else
    next_thread = 0;
}
```

测试结果



```
liudp5@ubuntu: ~/Desktop/xv6-riscv-fall19

xv6 kernel is booting

virtio disk init 0
hart 1 starting
hart 2 starting
init: starting sh
$ uthread
thread_a started
thread_b started
thread_c started
thread_c 0
thread_a 0
thread_b 0
thread_c 1
thread_a 1
thread_b 1
thread_c 2
thread_a 2
thread_b 2
thread_c 3
thread_a 3
thread_b 3
```



```
liudp5@ubuntu: ~/Desktop/xv6-riscv-fall19

thread_c 94
thread_a 94
thread_b 94
thread_c 95
thread_a 95
thread_b 95
thread_c 96
thread_a 96
thread_b 96
thread_c 97
thread_a 97
thread_b 97
thread_c 98
thread_a 98
thread_b 98
thread_c 99
thread_a 99
thread_b 99
thread_c: exit after 100
thread_a: exit after 100
thread_b: exit after 100
thread_schedule: no runnable threads
$
```

Xv6 lab: Lock/Memory allocator

由于有多个 CPU
实现一个多 free list 的初始化

```
struct kmem{
  struct spinlock lock;
  struct run *freelist;
} kmem;

struct kmem kmems[NCPU];//多个 kmem
void
kinit()//实现多 free list 的初始化
{
  push_off();
  int currentid = cpuid();
  pop_off();

  printf("# cpuId:%d \n",currentid);
  /* 初始化 NCPU 个锁*/
  for (int i = 0; i < NCPU; i++)
  {
    initlock(&kmems[i].lock, "kmem");
  }
  //initlock(&kmem.lock, "kmem");
  freerange(end, (void*)PHYSTOP);
}
```

实现插入和弹出，将 free list 的插入 push()和弹出 pop()操作封装为函数

```
struct run* trypopr(int id){
  struct run *r;
  r = kmems[id].freelist;
  if(r)
    kmems[id].freelist = r->next;
  return r;
}

void trypushr(int id, struct run* r){
  if(r){
    r->next = kmems[id].freelist;
    kmems[id].freelist = r;
  }
  else
  {
    panic("cannot push null run");
  }
}
```

修改 kfree()函数，查看空余 CPU，用 cpuid()获取相应的 id，然后将被释放的块插入相应的 free list

```
void
kfree(void *pa)
{
  struct run *r;

  if(((uint64)pa % PGSIZE) != 0 || (char*)pa < end || (uint64)pa >= PHY
STOP)
    panic("kfree");
  // Fill with junk to catch dangling refs.
  memset(pa, 1, PGSIZE);
  r = (struct run*)pa;

  push_off();
  int currentid = cpuid();
  pop_off();

  acquire(&kmems[currentid].lock);
  trypushr(currentid, r);
  release(&kmems[currentid].lock);
}
```

当一个 CPU 的 free list 为空，但是另一个 CPU 的 free list 还有空闲块时，我们就应该从有空闲 free list 的 CPU 处“偷”一个空闲内存块。

在 kalloc()函数中，首先检查当前 CPU 的 free list，如果当前 CPU 的 free list 不为空，那就直接 pop 一个 run 出来，将之初始化后，返回给调用者；如果当前 CPU 对应的 free list 为空，就需要去查询其它 CPU 对应的 free list，找到空闲的块 r 后，按照如下步骤操作：

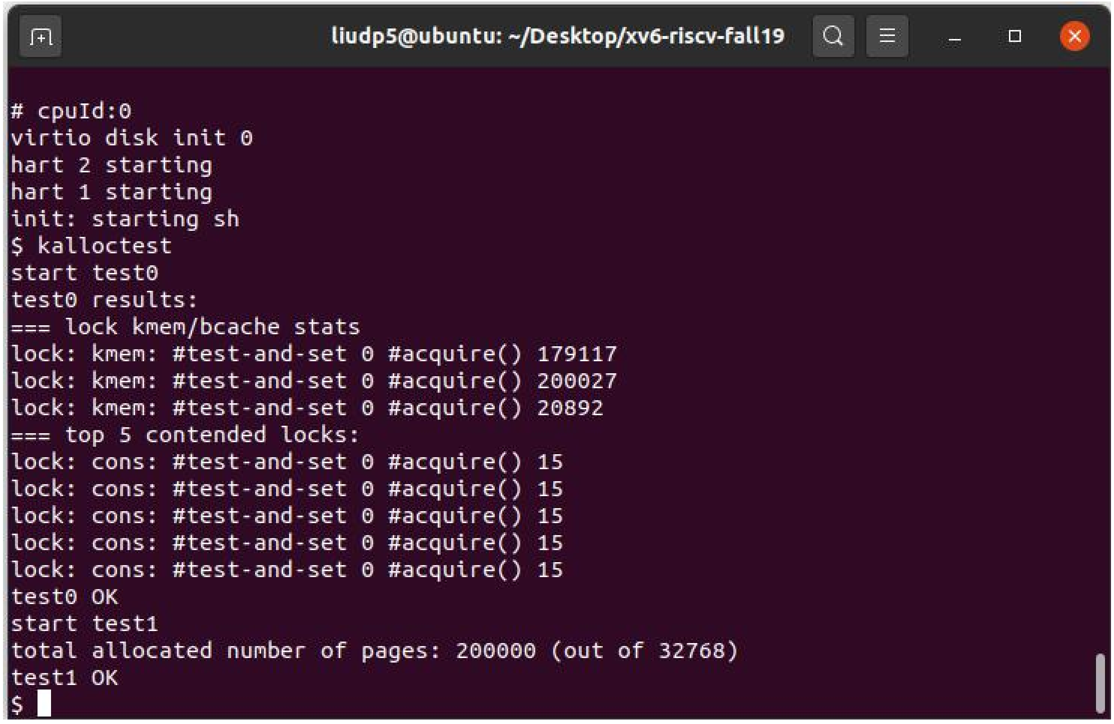
1.将 r 插入自己的 free list 中；

2.将 r 从自己的 free list 中弹出，将之初始化，返回给调用者；

```
void *
kalloc(void)
{
  struct run *r;
  int issteal = 0;/** 标识是否为偷盗 */
  push_off();
  int currentid = cpuid();
  pop_off();

  acquire(&kmems[currentid].lock);

  r = trypopr(currentid);
  /**
   * 将 id 的一块 free page 卸下，然后给 currentid
```

```
   * 这个过程中经历了：
   *
   * 1.卸下 id 的 free page
   * 2.为 current id 的 freelist 添加该 page
   * 3.将 current id 的 freelist 中的该 page 卸载掉
   * 4.返回该 page
   *
   * 整个过程完成了 current id 偷盗 id 的 free page 的行为
   */
  if(!r){
    //printf("oops out of memory\n");
    for (int id = 0; id < NCPU; id++)
    {
      /* steal first run */
      if(id != currentid){
        /** 锁住 id 的 freelist，此时不让其他 cpu 访问  */
        if(kmems[id].freelist){
          acquire(&kmems[id].lock);
          /** 卸下 id 的 free page */
          r = trypopr(id);
          /** 为 currentid 的 freelist 添加一个 run */
          trypushr(currentid, r);
          issteal = 1;
          release(&kmems[id].lock);
          break;
        }
      }
      //printf("\n");
    }
  }
  /** 如果是偷盗的，则把 currentid 的 freelist 释放出来  */
  if(issteal)
    r = trypopr(currentid);

  release(&kmems[currentid].lock);

  if(r){
    //printf("currentid: %d, r: %p\n", currentid, r);
    memset((char*)r, 5, PGSIZE); // fill with junk
  }
  /** 返回该 page  */
  //printf("issteal: %d \n", issteal);
  return (void*)r;
}
```

测试结果



Xv6 lab: Lock/Buffer cache

首先修改 bcache，以使其支持哈希桶结构：

```
struct {
  struct spinlock lock;
  struct buf buf[NBUF];              // block[30]
  /** 实则循环双向链表  */
  // Linked list of all buffers, through prev/next.
  // head.next is most recently used.
  //struct buf head;

  struct buf buckets[NBUKETS];
  struct spinlock bucketslock[NBUKETS];
} bcache;
```

初始化 bcache：

```
void
binit(void)
{
  struct buf *b;
  /** 在 head 头插入 b  */
  initlock(&bcache.lock, "bcache");
  for (int i = 0; i < NBUKETS; i++)
  {
```

```
    initlock(&bcache.bucketslock[i], "bcache.bucket");
    bcache.buckets[i].prev = &bcache.buckets[i];
    bcache.buckets[i].next = &bcache.buckets[i];
  }

  for (b = bcache.buf; b < bcache.buf + NBUF; b++)
  {
    int hash = getHb(b);
    b->time_stamp = ticks;
    b->next = bcache.buckets[hash].next;
    b->prev = &bcache.buckets[hash];
    initsleeplock(&b->lock, "buffer");
    bcache.buckets[hash].next->prev = b;
    bcache.buckets[hash].next = b;
  }
}
```

bget()，其实完成这一部分完全就是将原代码改写为支持哈希桶即可。需要注意的是：
在 bget 函数里，如果找到相应缓冲区的话，就简单返回就行，但如果没找到，就去其他 hash 桶里偷个来放自己所属的缓冲区里，如果别的 hash 桶里没有的话就报错

```
static struct buf*
bget(uint dev, uint blockno)
{
  int hash = getH(blockno);
  struct buf *b;
  /**
   *
   * My modification
   */
  acquire(&bcache.bucketslock[hash]);

  for(b = bcache.buckets[hash].next; b != &bcache.buckets[hash]; b = b-
>next){
    if(b->dev == dev && b->blockno == blockno){
      b->time_stamp = ticks;
      b->refcnt++;
      //printf("## end has \n");
      release(&bcache.bucketslock[hash]);
      acquiresleep(&b->lock);
      return b;
    }
  }
```

```c
  // If there is no cached buffer for the given sector, bget must make one, possibly reusing a buffer
  // that held a different sector. It scans the buffer list a second time, looking for a buffer that
  // is not in use
  // (b->refcnt = 0); any such buffer can be used. Bget edits the buffer metadata to record the new
  // device and sector number and acquires its sleep-lock. Note that the assignment b->valid = 0
  // ensures that bread will read the block data from disk rather than incorrectly using the buffer’s
  // previous contents.
  // Not cached; recycle an unused buffer.

  /**
   *
   * My modification
   */
  for (int i = 0; i < NBUKETS; i++)
  {
    if(i != hash){
      acquire(&bcache.bucketslock[i]);
      for(b = bcache.buckets[i].prev; b != &bcache.buckets[i]; b = b->prev){
        if(b->refcnt == 0){
          b->time_stamp = ticks;
          b->dev = dev;
          b->blockno = blockno;
          b->valid = 0;     //important
          b->refcnt = 1;

          /** 将 b 脱出  */
          b->next->prev = b->prev;
          b->prev->next = b->next;

          /** 将 b 接入  */
          b->next = bcache.buckets[hash].next;
          b->prev = &bcache.buckets[hash];
          bcache.buckets[hash].next->prev = b;
          bcache.buckets[hash].next = b;
          //printf("## end alloc: hash: %d, has: %d\n", hash,i);
          release(&bcache.bucketslock[i]);
          release(&bcache.bucketslock[hash]);
          acquiresleep(&b->lock);
```

```
          return b;
        }
      }
      release(&bcache.bucketslock[i]);
    }
  }
  panic("bget: no buffers");
}
```

实现 brelse()，不需要采用任何 lock，利用时间戳机制就能解决问题

```
void
brelse(struct buf *b)
{
  //printf("#---------------------------------------- brelse! ---------
-------------------------------\n");
  if(!holdingsleep(&b->lock))
    panic("brelse");

  releasesleep(&b->lock);
  int blockno = getHb(b);
  b->time_stamp = ticks;
  if(b->time_stamp == ticks){
    b->refcnt--;
    if(b->refcnt == 0){
      /** 将 b 脱出  */
      b->next->prev = b->prev;
      b->prev->next = b->next;

      /** 将 b 接入  */
      b->next = bcache.buckets[blockno].next;
      b->prev = &bcache.buckets[blockno];
      bcache.buckets[blockno].next->prev = b;
      bcache.buckets[blockno].next = b;
    }
  }
}
```

测试结果

```
hart 2 starting
hart 1 starting
init: starting sh
$ bcachetest
start test0
test0 results:
--- lock kmem/bcache stats
lock: kmem: #fetch-and-add 0 #acquire() 32982
lock: kmem: #fetch-and-add 0 #acquire() 59
lock: kmem: #fetch-and-add 0 #acquire() 39
lock: bcache: #fetch-and-add 0 #acquire() 2481
lock: bcache: #fetch-and-add 0 #acquire() 1448
lock: bcache: #fetch-and-add 0 #acquire() 2496
lock: bcache: #fetch-and-add 0 #acquire() 2170
lock: bcache: #fetch-and-add 0 #acquire() 3179
lock: bcache: #fetch-and-add 0 #acquire() 3176
lock: bcache: #fetch-and-add 0 #acquire() 3392
lock: bcache: #fetch-and-add 0 #acquire() 3219
lock: bcache: #fetch-and-add 0 #acquire() 4781
lock: bcache: #fetch-and-add 0 #acquire() 3601
lock: bcache: #fetch-and-add 0 #acquire() 2736
lock: bcache: #fetch-and-add 0 #acquire() 2475
lock: bcache: #fetch-and-add 0 #acquire() 1474
--- top 5 contended locks:
lock: virtio_disk: #fetch-and-add 191397 #acquire() 1378
lock: proc: #fetch-and-add 113049 #acquire() 87151
lock: proc: #fetch-and-add 56438 #acquire() 86747
lock: proc: #fetch-and-add 54043 #acquire() 86726
lock: proc: #fetch-and-add 48809 #acquire() 86726
tot= 0
test0: OK
start test1
test1 OK
```

Xv6 lab: File System/Large files

1. 为 inode 添加一个二级索引，使之能够支持分配更大的文件；
2. 学会如何创建 symbol link 软链接

为 inode 添加一个 Doubel Indirect 索引块，这样每一个 inode 就能够支持 11 + 256 + 256 * 256 = 65803 个数据块的文件了。修改宏定义如下

```
/** direct blocks */
#define NDIRECT 11
/** indirect blocks */
#define SINGLEDIRECT (BSIZE / sizeof(uint))
#define NINDIRECT (SINGLEDIRECT + SINGLEDIRECT * BSIZE / sizeof(uint))
#define MAXFILE (NDIRECT + NINDIRECT)
```

其中，NDIRECT 代表直接索引指向块的数量；SINGLEDIRECT 表示一级索引能够指向的块的数量；NINDIRECT 索引代表一级和二级索引一共可指向的块的数量。

修改 kernel/param.h 中的 FSSIZE 宏定义为 200000

```
#define FSSIZE       200000  // size of file system in blocks
```

模仿 bmap 原有分配块的操作，实现二级索引列表

```
static uint
bmap(struct inode *ip, uint bn)
{
  /**
   * bmap() deals with two kinds of block numbers.
   * The bn argument is a "logical block number" --
   * a block number within the file, relative to the start of the file.

   * The block numbers in ip->addrs[], and the argument to bread(),
   * are disk block numbers.
   * You can view bmap() as
   *
   *
   * mapping a file's logical block numbers into disk block numbers.
   */
  // printf("-----------------------------------------------\n");
  // printf("bn1: %d\n", bn);
  // NINDIRECT 256
  // printf("NINDIRECT: %d\n", NINDIRECT); //256
  uint addr, *a, *a2;
  struct buf *bp, *bp2;
  /** bn 是相对 file（inode）的虚拟编号  */
  /** 直接返回块 ip->addrs[bn]  */
  if(bn < NDIRECT){
    if((addr = ip->addrs[bn]) == 0)
      ip->addrs[bn] = addr = balloc(ip->dev);
    return addr;
  }
  /** 否则减去 NDIRENT  */
  bn -= NDIRECT;
  //11 + 256 + 256 * 256
  if(bn < NINDIRECT){
    if(bn < SINGLEDIRECT){
      // Load indirect block, allocating if necessary.
      if((addr = ip->addrs[NDIRECT]) == 0)
        ip->addrs[NDIRECT] = addr = balloc(ip->dev);
      /** Return a locked buf with the contents of the indicated block.
  */
      bp = bread(ip->dev, addr);
      /**
       * 为何 #define NINDIRECT (BSIZE / sizeof(uint)) ?
```

```
         * 因为一个 buf 有 BSIZE 个 data，一个 block 地址为 64 位，故刚好存储 256 个
block 指针，NINDIRECT = 256
         */
      /** 一个文件描述符 */
      a = (uint*)bp->data;
      if((addr = a[bn]) == 0){
        a[bn] = addr = balloc(ip->dev);
        log_write(bp);
      }
      brelse(bp);
    }
    else
    {
      /**
       * 实现 double-indirect
       * 结构:
       * addr -> 256 个 single-indirect -> 256 * 256 个 block
       * 256 + 11 = 267 0 ~ 266 为前面的，267 开始后面为后面的
       *  */
      bn -= SINGLEDIRECT;
      /** 宏在做乘除运算时记得加括号 5555555555555555555  */
      /** single-indirect 索引*/
      int single_indirect_index = bn / SINGLEDIRECT;
      /** single-indirect 内部相对索引  */
      int relative_offset_bn = bn % SINGLEDIRECT;
      /** 下标 12 是 double-indirect  */
      int pos = NDIRECT + 1;
      if((addr = ip->addrs[pos]) == 0)
        ip->addrs[pos] = addr = balloc(ip->dev);

      bp = bread(ip->dev, addr);
      a = (uint*)bp->data;
      if((addr = a[single_indirect_index]) == 0){
        printf("bn: %d, addr: %p\n",bn, addr);
        a[single_indirect_index] = addr = balloc(ip->dev);
        log_write(bp);
      }
      brelse(bp);
      bp2 = bread(ip->dev, addr);
      a2 = (uint*)bp2->data;
      if((addr = a2[relative_offset_bn]) == 0){
        a2[relative_offset_bn] = addr = balloc(ip->dev);
        log_write(bp2);
      }
```

```
      brelse(bp2);
    }
    printf("-----------------------------------------------\n");
    return addr;
  }
  panic("bmap: out of range");
}
```

将 itrunc 补充完整，以释放 inode 下的所有数据块，同样，做法参考原本的 itrunc 即可

```
static void
itrunc(struct inode *ip)
{
  int i, j;
  struct buf *bp, *bp2;
  uint *a, *a2;
  /** Free 掉所有 Direct */
  for(i = 0; i < NDIRECT; i++){
    if(ip->addrs[i]){
      bfree(ip->dev, ip->addrs[i]);
      ip->addrs[i] = 0;
    }
  }
  /** Free 掉 single-direct  */
  if(ip->addrs[NDIRECT]){
    bp = bread(ip->dev, ip->addrs[NDIRECT]);
    a = (uint*)bp->data;
    /**
     * 修改 NINDIRECT 为 SINGLEDIRECT
     * for(j = 0; j < NINDIRECT; j++){
     *   if(a[j])
     *     bfree(ip->dev, a[j]);
     * }
     */
    for (j = 0; j < SINGLEDIRECT; j++)
    {
      /* code */
      if(a[j])
        bfree(ip->dev, a[j]);
    }

    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT]);
    ip->addrs[NDIRECT] = 0;
  }
```

```
  /** Free 掉 double-direct  */
  if(ip->addrs[NDIRECT + 1]){
    printf("free double\n");
    int pos = NDIRECT + 1;
    bp = bread(ip->dev, ip->addrs[pos]);
    a = (uint*)bp->data;
    int number_of_single_direct = BSIZE / sizeof(uint);
    for (i = 0; i < number_of_single_direct; i++)
    {
      if(a[i]){
        bp2 = bread(ip->dev, a[i]);
        a2 = (uint *)bp2->data;
        for (j = 0; j < SINGLEDIRECT; j++)
        {
          if(a2[j])
            bfree(ip->dev, a2[j]);
        }
        brelse(bp2);
        bfree(ip->dev, a[i]);
        a[i] = 0;
      }
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[pos]);
    ip->addrs[pos] = 0;
  }
  ip->size = 0;
  iupdate(ip);
}
```

测试结果

```
$ bigfile
.....................................................................................................................................
.....................................................................................................................................
.....................................................................................................................................
.....................................................................................................................................
...................................................................................................................................
wrote 65803 blocks
bigfile done; ok
```